

Modern Approach

Feature + classification networks

- End-to-end, joint optimization
- □ Nothing fixed or hand crafted
- Learning based on data





General 2D Image Operations

✤ Point operation – some function of the pixel value
□ I'(x,y) = f(I(x,y)) or I'_{ij} = f(I_{ij})
□ Examples: log, sqrt, threshold

❖ Local area operation – some function of the pixel values in the area surrounding the pixel
 □ I'_{ij}=f({I_{(i+u)(j+v)}}) where -m<u<m and -n<v<n
 □ Examples: blur, low-pass, high-pass, gradient, center-surround

Global operation – some function of the whole image
 Examples: histogram, mean value, median value, 2nd moment



Point Operations











Point Operations







Global examples



Same histogram



- Histogram: Distribution (count) of possible image values
- Average pixel value



Same average value





Global examples



Indoors



Outdoors







Image neighborhoods

Q: What happens if we reshuffle all pixels within the images?



A: Its histogram won't change.Point-wise processing unaffected.

Need to measure properties relative to small *neighborhoods* of pixels



Comparison

Point operations and global operations don't tell much about the object and scene

- Objects/surfaces have finite area
- □ Most surfaces have texture (defined by a local area)
- □ This is referred to as "aperture problem"
- Point operations very small aperture
 - Only information of a *single* pixel is used, no other spatial or temporal information
- Global operations very large aperture
 - How do you focus on what is relevant?
 - How do you deal with computational complexity?
 - Again, human vision system can solve focusing and complexity issues, but not current machine vision systems



Area operations: Linear filtering

- Much of computer vision analysis starts with local area operations and then builds from there
 - Texture, edges, contours, shape, etc.
 - Perhaps at multiple scales (because how do you know how large should your aperture be?)
- Linear filtering is an important class of local operators
 - Convolution
 - Correlation
 - □ Fourier (and other) transforms
 - Sampling and aliasing issues



The response of a linear shift-invariant system can be described by the *convolution* operation



Think of 2D convolution as the following procedure

- For every pixel (i,j):
 - Line up the image at (i,j) with the filter kernel
 - □ Flip the kernel in both directions (vertical and horizontal)
 - □ Multiply and sum (dot product) to get output value R(i,j)



M - 1 N - 1**Convolution** $R_{ij} = \sum_{m=0} \sum_{n=0}^{\infty} H_{mn} F_{i-m,j-n}$

• For every (i,j) location in the output image R, there is a summation over the local area



So, what do you get if you convolve an image with



Convolution is a linear filter (a linear shift-invariant operation)

$$F * G = G * F$$
 $F * (G * H) = (F * G) * H$

Symmetric

Associative

What do you get if you convolve an image with



Differentiation

***** Because of associative property, $(I^*A)^*B = I^*(A^*B)$

□ So it is the same as convolving with this





✤ What about



It is the same as convolving with this

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

Border effects can be a bit tricky



Linear Filter: noise reduction





- We can measure **noise** in multiple images of the same static scene.
- How could we reduce the noise, i.e., give an estimate of the true intensities?



Gaussian noise



 $f(x,y) = \overbrace{\widehat{f(x,y)}}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}} \qquad \qquad \text{Gaussian i.i.d. ("white") noise:} \\ \eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

>> noise = randn(size(im)).*sigma;



>> output = im + noise;

sigma=1

Effect of sigma on Gaussian noise:

Image shows the noise values themselves.

New Jacobia

sigma=4



Effect of sigma on Gaussian

Image shows the noise values themselves.





Effect of sigma on Gaussian noise:

Image shows the noise values themselves.



Effect of sigma on Gaussian noise:

This shows the noise values added to the raw intensities of an image.





Effect of sigma on Gaussian noise

This shows the noise values added to the raw intensities of an image.



sigma=16

Motivation: noise reduction



- How could we reduce the noise, i.e., give an estimate of the true intensities?
- ***** What if there's only one image?



First attempt at a solution

- Let's replace each pixel with an average of all the values in its neighborhood
- Assumptions:
 - Expect pixels to be like their neighbors
 - Expect noise processes to be independent from pixel to pixel



First attempt at a solution

- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:





Weighted Moving Average

Can add weights to our moving average
Weights [1, 1, 1, 1, 1] / 5





Weighted Moving Average

Non-uniform weights [1, 4, 6, 4, 1] / 16





F[x, y]

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

G[x, y]





F[x, y]

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

G[x, y]





F[x, y]

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

G[x, y]





F[x, y]

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

G[x, y]

0	10	20	30			



F[x, y]

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

G[x, y]

0	10	20	30	30		



F[x, y]

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

G[x, y]

0	10	20	30	30	30	20	10	
0	20	40	60	60	60	40	20	
0	30	60	90	90	90	60	30	
0	30	50	80	80	90	60	30	
0	30	50	80	80	90	60	30	
0	20	30	50	50	60	40	20	
10	20	30	30	30	30	20	10	
10	10	10	0	0	0	0	0	



Averaging filter

What values belong in the kernel H for the moving average example?



 $G = H \otimes F$



Smoothing by averaging



depicts box filter: white = high value, black = low value





filtered



original
Gaussian filter

What if we want nearest neighboring pixels to have the most influence on the output?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
F[x, y]									

H[u, v]

This kernel is an approximation of a Gaussian function:

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2 + v^2}{\sigma^2}}$$





Smoothing with a Gaussian









Wider smoothing kernel \rightarrow

smoothing



σ=0.2

σ=1 pixel







Boundary issues

* What is the size of the output?

• MATLAB: filter2(g, f, *shape*)

shape = 'full': output size is sum of sizes of f and g

shape = 'same': output size is same as f

shape = 'valid': output size is difference of sizes of f and g



Boundary issues

What about near the edge?

- □ the filter window falls off the edge of the image
- need to extrapolate
- □ methods:
 - clip filter (black)
 - wrap around
 - > copy edge
 - reflect across edge





Boundary issues

What about near the edge?

□ the filter window falls off the edge of the image

need to extrapolate

methods (MATLAB):

> clip filter (black): imfilter(f, g, 0)

wrap around: imfilter(f, g, 'circular')

- > copy edge: imfilter(f, g, 'replicate')
- reflect across edge:

imfilter(f, g, 'symmetric')



Median filter



- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise



Median filter





Median filtered

Plots of a row of the image



Median filter

Median filter is edge preserving









Frequency Decomposition

Basis

- **☆** X, Y, Z
- $\clubsuit V = a X + b Y + c Z$
- Any vector can be decomposed into the bases





Any waveform can be decomposed into the Fourier bases



High vs. Low Frequencies

- Slow varying components
- Smooth regions
- Accentuated by suppressing the highfrequency component
- E.g., using box orGaussian smoothing

- Fast varying components
- Edges and noise
- Accentuated by suppressing the lowfrequency component
- E.g. using derivative operators



Gaussian filter

Symmetric Gaussian kernel:

Large σ

Small σ





Mean: (0,0)Standard deviation: σ



Gaussian filtering (smoothing, blurring) is a good model of camera optics

Gaussian filters

What parameters matter here?

Size of kernel or mask

Note, Gaussian function has infinite support, but discrete filters use finite kernels







Gaussian filters

*What parameters matter here?

Variance of Gaussian: determines extent of smoothing





Convolution

- For a high-pass filter, the kernel values should add to zero
 - Blocks out low spatial frequencies (gradual changes), accentuates high spatial frequencies (rapid changes)
 - Accentuates image noise!
- For a low-pass filter, the kernel values should add to one
 - Blocks out high spatial frequencies
 - Smoothing, averaging, blurring
 - Reduces image noise!







High pass



High-pass filter example



Low-pass Gaussian filter example



Correlation

Correlation is *almost* the same thing as convolution
Minus the "flip the kernel in both directions" step
So it's somewhat more intuitive than convolution

Normalized correlation is a frequently used operation in computer vision

□ More on that later...





All the discussions so far

- Filtering
- Convolution, correlation, etc.
- Average, mean, Gaussian
- ☐ High-pass, low-pass
- Lead us to edge detection which needs all the terminologies and techiques



Edge Detection

- Edge detection is a local area operator that seeks to find significant, meaningful changes in image intensity (color?) that correspond to
 - Boundaries of objects and patterns

Texture

- Changes in object color or brightness
- **Highlights**
- Occlusions
- Etc.





Example









Vertical edges

Edge magnitude





Edge detection examples





The bad news

Unfortunately, it's very hard to tell *significant* edges from *bogus* edges!

□ Noise is a big problem!

- An edge detector is basically a high-frequency filter, since sharp intensity changes are high-frequency events
- But image noise is also high-frequency, so edge detectors tend to accentuate noise!
- Some things to do:
 - Smooth before edge detection (hoping to get rid of noise but not edges!)
 - Look for edges at multiple scales (pyramids!)
 - Use an adaptive edge threshold



Caveats

In reality, low light levels and random noise lead to high fluctuations in individual pixel values, leading to bad estimations.





Graphically





Edge detection history

Edge detection has a long history and a huge literature

- Edge modeling: Step edges, roof edges, impulse edges...
- Biological modeling: How does human vision do it?
- Elegant and complex mathematical models
- Simple and computationally cheap edge detectors
- □ Etc., etc., etc....
- Typical usage:
 - Detect "edge points" in the image (filter then threshold)
 - > Edges may have magnitude **and** orientation
 - Throw away "bad" ones (isolated points)
 - Link edge points together to make edge segments
 - □ Merge segments into lines, corners, junctions, etc.
 - □ Interpret these higher-level features in the context of the problem



Edge detection

The bottom line:

- ☐ It doesn't work!
- At least, now how we'd like it to:
 - > Too many false positives (noise)
 - > Too many omissions (little or no local signal)
- Still, edge detection is often the first step in a computer vision program
 - U We have to learn to live with imperfection



How to design an edge detector?



What should an edge detector output?

Where does the edge exist?















Edge detection: Three primary steps

- Smoothing
 - Reduce noise
 - Reduce frequency content not of interest
 - Are we looking for low, medium, or high frequency edges?

Edge enhancement

- Filter (usually linear) to produce high values where there are strong edges, low values elsewhere
- Edge localization
 - Where specifically are the edge points?





Edge detection criteria

Optimal edge detector for competing criteria:

- Detection
 - Maximize detection (minimize misses (false negatives))
 - Minimize false positives
- Localization
 - Location of detected edge should be as close as possible to true location
- There is a tradeoff between these two criteria
 - Compromise between good detection and good localization for a given approach

How can you assure no misses?

How can you assure no false positives?







Canny edge detector

Properties of the Canny edge detector

- Designed for step edges (but generalizes relatively well)
- Good detection
- Good localization
- Single response constraint
 - Uses *nonmaximum supression* to return just one edge point per "true" edge point
 Avoids this:



- Basic detection filter:
 - ID first derivative of a Gaussian
 - Apply this at multiple orientations to detect 2D edges and their orientations



Example of measuring at multiple orientations



Canny steps



- Apply Gaussian smoothing
 - □ 2D Gaussian filter over the whole image
 - Can actually be done with H and V 1D filters (much faster)
- Run 1D detectors at multiple orientations
 Produce *edge strength* and *edge orientation* maps
- Run nonmaximum suppression to eliminate extra edge points, producing only one per true edge point (ideally)
- Threshold to produce a binary (*edge* or *no edge*) edge image
 - Alter the threshold value depending on local edge information




original image (Lena)





Combine the 1D detectors → Edge strength map





Thresholding





Thinning (non-maximum suppression)



Image gradient

* The gradient of an image:
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid change in intensity

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x}, 0 \end{bmatrix}$$

$$\nabla f = \begin{bmatrix} 0, \frac{\partial f}{\partial y} \end{bmatrix}$$

$$\nabla f = \begin{bmatrix} 0, \frac{\partial f}{\partial y} \end{bmatrix}$$

The gradient direction is given by:

$$\theta = \tan^{-1} \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$$

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$



The discrete gradient

How can we differentiate a *digital* image f[x,y]?
Option 1: reconstruct a continuous image, then take gradient
Option 2: take discrete derivative (finite difference)

$$\frac{\partial f}{\partial x}[x,y] \approx f[x+1,y] - f[x,y]$$

How would you implement this as a correlation?







Other (simpler) edge detector

Sobel detector





Prewitt detector





 $|G| = G_x^2 + G_y^2$ $\theta = \operatorname{atan} G_y / G_x$







Other Possibilities

8-directional masks





Sobel example

Original







Vertical edges

Edge magnitude





More edge detectors









Etc.

Edge detectors are not limited to 3x3 kernels



Multiple Scales

Bad localization

Where is the edge?





Multiple Scales (cont.)



Bad localization

Where is the edge?



Sharp vs. Fuzzy







Fig. 2. The problem of local estimation scale. Different structures in a natural image require different spatial scales for local estimation. (a) The original image contains edges over a broad range of contrasts and blur scales. (b) The edges detected with a Canny/Deriche operator tuned to detect structure in the mannequin. (c) The edges detected with a Canny/Deriche operator tuned to detect the smooth contour of the shadow. Parameters are ($\alpha = 1.25$, $\omega = 0.02$) and ($\alpha = 0.5$, $\omega = 0.02$), respectively. See [2] for details of the Deriche detector.



Response to a blurred step edge



$$\sigma_b = \sqrt{\left(d/2^2\right) - \sigma_2^2}$$

Thus, the blur due to defocus can be estimated from the measured thickness of the contours, after compensation for the blur induced by the estimation itself.







(a)





Fig. 10. Depth segmentation based on focal blur. (a) A photograph of tree branches with shallow depth of field (#3.5) and near focus. (b) Edge map. (c) Foreground structure (focused contours). (d) Background structure (blurred contours).



Multiple Scales (cont.)

Basically, considering your operator myopic – that it uses only information in a small neighborhood of a pixel

♦Why?

Saving in computation effort

□Local coherency

*But how large should the neighborhood be?

- □Too small not enough information
- □Too large Too noisy, with multiple edges



Multiple Scales (cont.)

No single scale will suffice

- E.g. sharply focused objects have fast transition edges, out-of-focus objects have slow transition edges
- Need operators that are tuned to edges of different scales
- For an operator with a large processing window, the content can be noisy (multiple edge presence)
 - □Need a mechanism to smooth out small edges



2D edge detection filters



 ∇^2 is the **Laplacian** operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$



Multiple Scales





Multiple Scales







1 pixel

3 pixels

7 pixels

The scale of the smoothing filter affects derivative estimates, and also the semantics of the edges recovered.





We wish to mark points along the curve where the magnitude is biggest. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?



q Gradient \cap ľ

Non-maximum suppression

At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.





Predicting the next edge point Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either



r or s).







fine scale high threshold













